MOTOROLA BROADBAND
COMMUNICATIONS SECTOR
Intellectual Property Department
For Internal Use Only

Invention Record Form

BCS Docket No. D3043

RECEIVED
SEP 12 2002
GENERAL INSTRUMENT CORPORATION
INTELLECTUAL PROPERTY

I. Administrative Information

1. Short Descriptive Title of the Invention:

Method For Authenticated Storage outside a Security Processor

2. Identify all persons who contributed to this invention, including persons from other divisions and/or outside companies:

	Inventor 1	Inventor 2
Full Legal Name	Eric Sprunk	Paul Moroney
Home Address	7309 Bolero Street	3411 Western Springs Rd
City, State, Zip	Carlsbad, CA 92009	Encinitas, CA 92024
Citizenship	USA	USA
Division/Co. Location	Advanced Technology Motorola BCS	Advanced Technology Motorola BCS
Office Phone No.	858-404-2426	858-404-2446
Mgr.'s Name & Phone No.	Paul Moroney 858-404-2446	Carl McGrath 215-323-1412
Signature of Inventor	[illegible]	[illegible]
Date	8/13/02	9-10-2002

	Inventor 3	Inventor 4
Full Legal Name		
Home Address		
City, State, Zip		
Citizenship		
Division/Co. Location		
Office Phone No.		
Mgr.'s Name & Phone No.		
Signature of Inventor		
Date		

3. ☐ Check box if there are additional inventors listed on separate sheets. Additional information concerning inventors, if any.

Invention Record Form

II. Background Information

1. Do you believe this invention was developed while working under or in the performance of experimental, developmental or research work called for by a government contract or with the understanding that a government contract would be awarded? [✓ No] ☐ Yes If yes, please explain:
2. Has your invention been disclosed to anyone outside Motorola in a speech, exhibit, presentation, product, product manual, report, lecture, trade show, technical article, publication or otherwise? [✓ No] ☐ Yes If yes, please explain:
3. Is this invention related to any previous Motorola invention disclosures of which you are aware (made by you or someone else)? [✓ No] ☐ Yes If yes, please explain:
4. Name of product(s) and/or project(s) for which this invention was developed:

   DRM (Digital Rights Management) and Copy Protection Systems of the future
5. Planned or actual use of invention:

   In security processors following the MC2.1.
6. What economic benefits do you think Motorola can derive from this invention?

   [redacted]
7. When do you expect a product incorporating this invention to be sold, offered for sale or shown to someone outside of Motorola? (If a product or prototype has already been sold, offered for sale or shown, please identify the earliest date this happened.)

   During 2003.
8. Has a working model of the invention been built and tested (or appropriate software been written)? [✓ No] ☐ Yes If yes, who has witnessed a demonstration, and when?
9. List below any patents, publications, articles, texts, products, etc. which describe technology similar to your invention including reference material which may be useful in understanding the background technology of your invention. (Use a separate sheet if necessary and attach a copy of each item. Please include copies of all bibliographical information.) (Use a separate sheet if necessary)

   None known.

[illegible] 8/13/02 [illegible] 9/10

Signature of Submitter(s)

Date

[illegible]

Read and understood by [Witness Signature(s)]

9-10-02

Date

Invention Record Form

III. Description of the Invention

1. Please provide a very brief (i.e., one short sentence) summary of your invention.

   A method for securely storing large amounts of data outside a memory-limited security processor.

2. Briefly describe the field of technology to which your invention relates.

   Digital Rights Management, Conditional Access, and Copy Protection

3. Briefly describe the problems, issues or needs which led to the invention

   A security processor (SP) ASIC has secure internal memory for keys, etc., but may need to keep large amounts of secure data beyond its memory capacity. There are inherent security challenges in being certain that externally stored data is not manipulated by a pirate, and this invention solves those challenges.

4. How have others addressed these problems, issues or needs?

   I am not aware of anyone trying to solve this problem.

5. Describe those particular features or functions of your invention which you think may be novel or technical advancements over the technology you listed in section II.9.

   Since I have seen no other solution to this problem it seems to be novel.

6. Best Mode: Describe any and all preferences you personally have regarding how to best implement, build, produce or use your invention (e.g., preferred parts, materials, techniques, etc. which you feel are best in practicing your invention). Each submitter's opinion is important here, even if there is disagreement. Please list anything you think will make the invention better in any way.

   See attached.

7. Briefly describe any alternative uses, variations or modifications of your invention which you contemplate.

   See attached.

8. Please provide any additional information you think should be known by the attorney reviewing this form.

9. Please provide a detailed description of your invention. Your description should ideally provide as many details of your invention as possible in order to achieve optimal patent protection. An ideal disclosure should describe the construction and operation of the invention including drawings (flow charts, schematics, block diagrams, mechanical drawings, photographs, etc.) and any relevant engineering laboratory notebook pages, reports, program listings, etc. If you have already prepared reports or other descriptive information, there is no need to rewrite it. Simply attach it and reference it in your invention disclosure data sheet (for example, "see attached 9 page engineering progress report addressed to John Doe dated 1 Jan., 1992 for description of amplifier circuit").

   See attached.

[illegible] 8/13/02 [illegible] 9-10-02

Signature of Submitter(s) Date

Alexander Medvinsky 9-10-02

Read and understood by [Witness Signature(s)] Date

MOTOROLA CONFIDENTIAL & PROPRIETARY

(Part with Sept 19 2001 email at bottom)

From: Sprunk, Eric (SD-EX)
Sent: Thursday, September 20, 2001 10:58 AM
To: Moroney, Paul (SD-EX)
Subject: RE: gotta branch, gotta hash

I see, yes. There are some interesting minor implications, but it seems to work.

I think the MC must be able to run 2 hashes at once over the items starting with the change point. One is the recheck of the old data including the unedited ECDS, while the other is the new hash of the entire chain including the new edited ECDS. Our hash hardware allows us to suspend a hash in the middle, store its 64 byte state, do a different hash, restore the state of the original hash, and continue. Unfortunately, this is only possible with relatively slow firmware intervention, and not using the PCI DMA capability that can hash at max speed. We could add double hash state switching and tracking capability to the 2.2 SHA engine, but that is more VLSI work.

The concept of hash also requires adjustment. SHA has an init process that interferes with storing intermediate state to do multiple hashes at once. For simplicity, assume this init does not exist. Then, it boils down to be a real simple process: (again assume the Xth ECDS of N is edited)

- Hash ECDS 1 to ECDS X-1; we call this result the PREMAC.
- Thereafter run 2 hashes at once
    - one where the edited Xth ECDS is next to hash. Call this one the NEW CHAIN.
    - one where the unedited Xth ECDS is next. Call this one the OLD CHAIN.
- When done, the 2 results are called the NEWMAC and OLDMAC.
    - OLDMAC must match the MAC previously stored in secure RAM, else abort.
    - NEWMAC replaces OLDMAC if OLDMAC checked out above.

For completeness, I must point out that hybrid schemes exist. If we form a fixed tree of say, 4 leaves, then we can brute force over 1/4 the chain and get the log advantage on the rest.

The brute force method seems to require 1 hash of the entire chain (including a simultaneous hash of part of the chain) in a simple fashion, while the BTC method requires ~ LOG2(chain length) hashes every time, at greater complexity. Soooo, there is a complexity vs data volume tradeoff here.

I will now swag some data volume (i.e. number of ECDS's) estimates. Assume the following:

- PVRs disks double in size each year.
- Assume that P% of the disk space is taken up with movies where:
    - P = 90% for average user
    - P = 10% for power music user
- Each compressed song is 5MB.

year	size	avg user songs	power user songs
2002	80G	1.6K	14.4K
2003	160G	3.2K	28.8K
2004	320G	6.4K	57.6K
2005	640G	12.8K	115K
2006	1.28Terabyte	25.6K	230K

Clearly, the only thing that will bound the number of ECDS's is whether people want to use a PVR to store their music! I am not a power music user, and I think I have about 250 CDs, at 15 songs each that's 3750 songs. My brother is a power user, and he easily has 10 times that, maybe 100 times that. This surfaces a fundamental assumption I have been making for persistent DRM: everything has an ECDS, even if free or input locally by the consumer.

[illegible] 8/13/02 [illegible] 9-20-02

This assumption was made to avoid the PVR being useful for stolen content, and it has some limited effectiveness for that purpose. Even if we give locally-input content an ECDS (which says that is what it is), we still cannot truly identify that content in an inherent sense. Someone could put in the Rolling Stones and say it is their child singing; we'd never know. This suggests, however, that the titling and indexing of music must link to a title placed in the ECDS when the content was entered into the PVR. This way, if someone misidentifies content the false name is stuck in the PVR, and they are forced to use the false name to select and play that stolen content. Further, we could theoretically upload the titles of all songs for scanning, if the content owners so demanded. Hmmmm, I guess I still think 100% of content should have an ECDS.

This last point only partly affects the problem anyway. We must plan for success where the system has sold and downloaded lots of movies and songs to the user, and thus created lots of ECDS's. In a success scenario many thousands of ECDS's seems quite plausible. This raises the hard to answer question of exactly where the brute force scheme dies and the BTC scheme becomes mandatory.

I think we need a focus group or something!

-----Original Message-----

From:	Moroney, Paul (SD-EX)
Sent:	Wednesday, September 19, 2001 3:01 PM
To:	Sprunk, Eric (SD-EX)
Subject:	RE: gotta branch, gotta hash

I am not sure I agree, entirely.

Once you have a brute force tree that is correct, and an authenticator and count that matches,
you need not read in the whole thing in order to change one thing.

You know where you are to edit, so you start by reading in whole thing to check it. While doing so,
you write down partial hash on stuff prior to where you change. Upon doing the change, you now do not need to
read the stuff back in. No way to harm it. However, I do see that you cannot do exactly the same trick for the items after the change point! You would however be able to construct a partial hash over them, while doing the input check.

when you read them in the second time to form the new overall hash, you must also compute the partial hash,
and check it upon completion. If it fails due to a hack, you reject the new item.

paul

hmmmmmmmmmmmm

-----Original Message-----

From:	~~Sprunk, Eric (SD-EX)~~
Sent:	Wednesday, September 19, 2001 1:57 PM
To:	~~Moroney, Paul (SD-EX)~~
Subject:	gotta branch, gotta hash

[illegible] 8/13/02 [illegible] 9-10-02

RE: BTC for external storage, I think there is a vulnerability if we do brute force hashing, as follows.

First, assume the following:

- The info set to hash (a group of X ECDS's) is too big to store inside the MC.
- Only the one that changed (ECDS N) is within the MC, right after it was edited by the MC. The older version is still outside the MC with the other X-1 ECDS's.
- The Nth ECDS that changed is a random one, neither the first nor last in the hash operation performed over the entire set of X ECDS's.

Step 1. The hashing must begin on ECDS 1 outside the MC, and proceed up to the Xth one that changed. Ergo, the MC loads in X-1 ECDS's, hashing them as it goes.

Step 2. The hashing proceeds over the Xth ECDS stored in the MC.

Step 3. The hashing resumes on ECDS X+1 by loading in the remaining N-X ECDS's from outside. The end result is a hash of the entire set.

The problem should be apparent already, yes? In Step 1, during the hashing process, there is no way for the MC to know that ECDS 1 to X-1 are still authentic. An attacker can tweak them during this process, and the changes will be reflected in the final result quite happily. This is because the MC does not store any authenticators that are specific to one ECDS, but not others, since this type of "ECDS level resolution" of authenticators is necessary to know if any have changed.

This problem likewise exists in Step 3.

The MC must store data in a strange way that, to me, heavily implies BTC. The data must have segments (e.g. X ECDS's) and authenticators, where the authenticators allow any random segment to change, while still completely authenticating the rest of them. One authenticator per segment / ECDS is one way to do it, but suffers from scaleability. Storing a tree is the same approach made efficient enough to scale, through its log characteristic.

Now examine the ECDS update procedure if BTC is used:

- 1. MC edits a random ECDS X of N in secure memory, but keeps a temp copy of its old hash.
- 2. MC loads (requests) LOG2(N) BKs from outside.
- 3. MC verifies these BKs using the temp copy of old hash, and current stored Root. If bad, reject BKs.
- 4. MC hashes ECDS X and the BKs to form new BKs and Root.
- 5. MC outputs the new BKs.
- 6. MC stores the new Root.
- 7. MC increments the BTC sequence counter.
- 8. MC outputs the BTC seq ctr.

This is not similarly vulnerable. Only the edited ECDS is ever even hashed anew. All others are authenticated through verified BKs, and the new BK values are secured into the internally-stored Root immediately. No one can submit fake ECDS data, I think.

Tweaks are possible, along the line of our previous discussion. Less than all BKs could be stored, so long as they are always checked in Step 3, and the data segment being hashed as a leaf of the tree need not be only one ECDS. Note also that the sequence counter is now serving only an administrative function for version tracking of the whole tree.

[illegible] 8/13/02 [illegible] 9-10-02

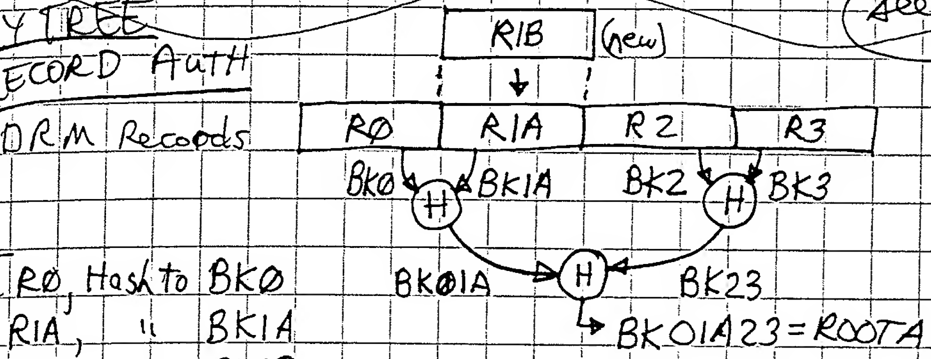
TITLE PROJECT N BOOK NO.

Work continued from Page —

~~Robin LaVoie (Cogeco) — need legal ref questions about SA breach, from Elroy~~ ~~Steve P.~~

(see 9/20/2001 email)

~~BINARY TREE~~ DRM RECORD AUTH



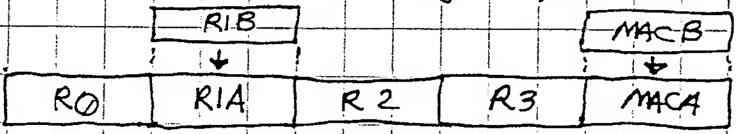
(all BKs stored outside MC in encrypted form)

(MC = security chip)

Steps

1. Load RØ, Hash to BKØ
2. " RIA, " BKIA
3. RIB, " BKIB
4. " BKØIA, confirm ≡ Hash(BKØ, BKIA)
   if not confirm abort, else continue (∴ RØ, BKØ are good)
5. Hash to BKØIB = Hash(BKØ, BKIB)
6. Encrypt " , store outside MC
7. Load BK23 and ROOTA
8. Confirm ROOTA ≡ Hash(BKØIA, BK23)
   if not confirm abort, else continue (∴ BK23 is good)
9. Form ROOTB = Hash(BKØIB, BK23)
10. Encrypt " , store outside MC.

LINEAR HASH DRM RECORD AUTH



HASHING: MACA = HASH(RØ→RIA→R2→R3)
MACB = " (RØ→RIB→R2→R3)
HØ = Hash(RØ only)
HIA = " (RØ→RIA)
HIB = " (RØ→RIB)
H2A = " (RØ→RIA→R2)
H2B = " (RØ→RIB→R2)
H3A = " (RØ→RIA→R2→R3)
H3B = " (" RIB " ")

1. Load RØ, Hash, suspend @ HØ.
2. Fork out into 2 Hashes from HØ:
   a) hash in RIA ⇒ HIA
   b) " " RIB ⇒ HIB
3. Continue both Hash chains:
   a) hash HIA→R2→R3 ⇒ H3A
   b) " HIB " " H3B
4. Confirm fork A: Is H3A ≡ MACA ?
   if No, abort, else continue (∴ RØ, R2, R3 good)
5. Encrypt H3B and store as new MACB.

SCIENTIFIC BINDERY PRODUCTIONS CHICAGO 60605 MADE IN USA

Work continued to Page

SIGNATURE [illegible] DATE 8/8/2002

DISCLOSED TO AND UNDERSTOOD BY DATE WITNESS DATE